Tampereen yliopisto

Vivian Lunnikivi

AUTOMAATTINEN ARVIOINTI OPETUSKÄYTÖSSÄ OHJELMOINNIN PERUSKURSSEILLA

Yhteenveto lähestymistavoista ja tapaustutkimus

Informaatioteknologian ja viestinnän tiedekunta
Kandidaatintutkielma
6/2019

TIIVISTELMÄ

Vivian Lunnikivi: Automaattinen arviointi opetuskäytössä ohjelmoinnin peruskursseilla
Kandidaatintutkielma
Tampereen yliopisto
Tieto- ja sähkötekniikka, tekniikan kandidaatti, tietotekniikka
Kesäkuu 2019

Ohjelmointi on muuttumassa yhdeksi digitalisoituvan yhteiskunnan kysytyimmistä taidoista. Samaan aikaan ohjelmoinnin osaajia on liian vähän. Kysynnän kasvaessa ohjelmoinnin opetukselle, kohdistuu opetuksen tarjoajille painetta kasvattaa kurssikokoja, mikä tuottaa lisää arviointityötä. Kurssihenkilökunnalle varatut resurssit ovat kuitenkin rajallisia, joten työtaakkaa helpottamaan on kehitetty automaattisia arviointijärjestelmiä.

Tässä työssä luodaan katsaus olemassa oleviin automaattisen arvioinnin menetelmiin, mitä arviointijärjestelmien kehittäjät voivat hyödyntää työssään. Työssä tunnistetaan ohjelmoinnin alkeiskursseille soveltuvia tehtävätyyppejä, kategorisoidaan palautteen laatua ja näkökulmia testaukseen, sekä vedetään yhteen testausstrategioita. Löydösten perusteella arvioidaan Tampereen yliopistolla ohjelmoinnin alkeiskurssilla käytössä olevaa arviointijärjestelmää.

Järjestelmän todetaan olevan ohjelmoinnin opetukseen soveltuva ympäristö. Järjestelmän käytöllä arviointityötä saadaan automatisoitua, joka mahdollistaa suuren tehtävien ja palautteen määrän, palautteen välittömyyden ja kurssin skaalautuvuuden suurillekin opiskelijamassoille. Järjestelmästä tunnistetaan kirjallisuusselvityksen perusteella kuitenkin myös kehityskohteita, kuten tyylintarkastamisen puuttuminen.

Avainsanat: Automaattiset tarkastimet, automaattinen arviointi, oppimisjärjestelmät, MOOC, ohjelmoinnin opetus, automatisoitu palaute

ABSTRACT

Vivian Lunnikivi: Educational Use of Automatic Assessment in Basic Courses of Programming
Bachelor's thesis
Tampere University
Computing and Electrical Engineering, Bachelor of Science
February 2019

Programming is becoming more and more sought-after skill in the digitalizing world. At the same time, there are too few skilled programmers. As the demand for programming education increases, so does the education providers' pressure to increment course sizes. This, however, produces more work in grading. Yet, as course personnel resources are limited, automatic grading systems are being developed to ease the workload.

In this paper, automatic assessment is summarized and discussed to provide an enumeration of existing approaches to help developers utilize these practices in their work. The conducted research identifies main categories of exercise types suitable for novice programming courses, categorizes feedback types and testing approaches, as well as summarizes testing strategies.

These findings are used as a basis for validating and finding improvement points from an existing automatic assessment system used on a basic programming course at the University of Tampere. The system is concluded to be suitable learning environment for a programming course, the use of which enables great amount of exercises and feedback, instant feedback and scalability of the course for large number of students. However, improvement points are identified from the system as well, as the system doesn't support style checking, for instance.

Keywords: Automatic assessment, LMS, Learning Management Systems, MOOC, programming education.

SISÄLLYSLUETTELO

1. JOHDANTO

Ohjelmointi on taito, jonka kysyntä kasvaa jatkuvasti digitalisoituvan maailman myötä. Ohjelmointia pidetään kuitenkin hankalana taitona oppia, ja se vaatiikin huomattavan määrän harjoittelua (1). Harjoittelun lisäksi ohjelmointiratkaisuista annetun palautteen on osoitettu parantavan oppimistuloksia (2).

Lisääntynyt tarve ohjelmoinnin osaajille näkyy korkeakouluille ja muille koulutuksen tarjoajille paineena tarjota ohjelmoinnin opetusta entistä suuremmalle määrälle opiskelijoita. Kurssikokojen kasvaessa myös arviointityö lisääntyy. Opiskelijat saattavat yksittäisellä ohjelmointikurssilla ratkaista kymmeniä tai jopa satoja ohjelmointitehtäviä, joiden arvioiminen käsin vaatii huomattavia opetusresursseja. (3)

Arviointiprosessia voidaan kuitenkin tehostaa automaattisen arvioinnin menetelmillä (4). Automaattiset tarkastimet tuottavat palautteen välittömästi, tarkastaminen on objektiivista sekä standardoitua, ja kerran toteutetulla tarkastimella voidaan nopeasti ja kustannustehokkaasti arvioida käytännössä rajaton määrä ratkaisuja samaan tehtävään. Näiden etujen saavuttamisen myötä opetuksentarjoajat voivat järjestää kursseja suurille opiskelijamäärille, samalla mahdollistaen kuitenkin riittävän määrän harjoituksia ja palautetta jokaiselle yksittäiselle opiskelijalle. (3)

Tampereen yliopisto on yksi monista ohjelmoinnin opetusta tarjoavista tahoista. Osalla yliopiston perusohjelmointikursseista on käytössä automaattisen arvioinnin järjestelmä, jota hyödynnetään osana opiskelijoiden palauttamien ohjelmointitehtävien palautteenantoa ja pisteytystä.

Tässä työssä luodaan katsaus automaattisen arvioinnin menetelmiin ja yhteenvetoon nojaten arvioidaan Tampereen yliopiston kurssikäytössä olevaa automaattista arviointijärjestelmää. Työssä arvioidaan järjestelmän soveltuvuutta kurssikäyttöön ohjelmoinnin johdantokurssilla ja järjestelmästä tunnistetaan kehityskohteita.

Luvussa 2 pohditaan aluksi oppimisen muotoja ja luodaan lyhyt katsaus verkkopohjaisiin oppimisjärjestelmiin. Luku 3 tekee yhteenvedon järjestelmien yleisesti hyödyntämistä tehtävätyypeistä ja niiden arvosteluun liittyvistä periaatteista. Luku 4 käsittelee yleisesti teknisiä lähestymistapoja tarkastimien toteuttamiseen ja luvussa 5 tarkastellaan miten tarkastimet on toteutettu eräällä Tampereen yliopiston ohjelmoinnin johdantokurssilla

Plussa-nimisessä oppimisjärjestelmässä. Järjestelmän toteutusta arvioidaan luvussa 6 ja yhteenveto työn tuloksista löytyy luvusta 7.

2. OPPIMINEN JA OPPIMISJÄRJESTELMÄT

Oppiminen on opiskelijan omaa, oppimiseen tähtäävää toimintaa, johon opetus vaikuttaa välillisesti ohjaamalla opiskelijan vuorovaikutusta opittavien asioiden kanssa. Suurin osa oppimisesta tapahtuu opiskelijan opiskellessa itsenäisesti. Siksi modernin oppimiskäsityksen mukaan aktiiviset oppimismenetelmät, kuten tekemällä oppiminen, tukevat oppimista paremmin verrattuna passiivisiin menetelmiin – esimerkiksi luennointiin. (5)

Opetusmenetelmien lisäksi myös arvioinnilla on keskeinen rooli oppimisessa. Opiskelijat kohdentavat opiskelunsa asioihin, joiden he olettavat painottuvan arvioinnissa. Siksi onkin tärkeää, että arviointi kohdistuu asioiden syvällistä oppimista tukeviin aktiviteetteihin ja on linjassa oppimistavoitteiden kanssa. Näin myös arviointi ohjaa oppimiseen. (5)

Arvioinnilla on myös muitakin oppimista tukevia rooleja. Arviointi tähtää selvittämään opiskelijan taitotasoa ja edistymistä kurssilla. Arvioinnin kautta opiskelijalle voidaan tuottaa kehittävää palautetta, jonka pohjalta opiskelija kykenee tunnistamaan puutteita ja virhekäsityksiä tiedoissaan ja toisaalta seuraamaan oppimistaan. Arvioinnilla vaikutetaan myös opiskelijoiden motivaatioon, ja erityisesti tehdyn työn huomioiminen arvioinnissa ja arvostelussa motivoi opiskelijaa työstämään kurssin aiheita. Parhaiden oppimistulosten saavuttamiseksi työn – ja siten myös arvioinnin – tulisi jakautua tasaisesti pitkin kurssia, sillä esimerkiksi pelkkään tenttiin perustuva arviointi rohkaisee opiskelijaa opiskelemaan vain juuri ennen tenttiä. (5)

Oppimisjärjestelmät (engl. *Learning Management Systems, LMS*) ovat oppimisalustoiksi tarkoitettuja digitaalisia palveluita, joiden yleisimpiin ominaisuuksiin kuuluvat muun muassa käyttäjänhallinta, kurssi-ilmoittautumisten käsittely, oppimateriaalin ja tehtävänantojen jakelu sekä palautuslaatikot tehtäville. (6) Massiivisista avoimista verkkokursseista (engl. *Massive Open Online Courses, MOOC*) puhutaan, mikäli oppimisjärjestelmä on toteutettu verkkopohjaisena avoimena palveluna ja tehtävien arvostelu hoidetaan palautuslaatikkoihin integroiduilla automaattisilla tarkastimilla. Kurssikokoja ei MOOC-kursseilla rajoiteta, jolloin samalle kurssille voi samanaikaisesti osallistua satoja tai jopa tuhansia opiskelijoita. (7)

MOOCit tarjoavat perinteisiin kursseihin verrattuna hyötyjä, kuten mahdollisuuden suorittaa kurssin etänä ja tuottaa enemmän palautetta palkkaamatta enempää kurssihenkilökuntaa tekemään arviointityötä. Arviointijärjestelmät pitävät kirjaa myös pisteistä sekä tehtävien määräajoista. Täten MOOCit mahdollistavat suuremmat opiskelijamäärät ja opetuksen tarjoamisen muillekin kuin koulutuksen tarjoajatahon omille opiskelijoille. (7)

Läpi MOOCien historian tutkimuksen ja kehitystyön tärkeimpiä kysymyksiä ovat olleet miten arvioida opiskelijan osaamista todenmukaisesti sekä miten tuottaa hyödyllistä ja havainnollista palautetta. Teknisestä näkökulmasta tarkasteltuna yksinkertaisimmat ratkaisut voidaan toteuttaa osana oppimisjärjestelmää siinä missä monimutkaisemmat arviointijärjestelmät toteutetaan kolmannen osapuolen palveluina, jotka yhdistetään oppimisjärjestelmään.

3. YLEISET TEHTÄVÄTYYPIT JA TEHTÄVIEN ARVIOINTI

Tämä luku käsittelee lyhyesti MOOCeissa usein esiintyviä tehtävätyyppejä ja tarkastelee niiden arvioimista automaattisen arvioinnin näkökulmasta. Ensimmäinen alaluku käy läpi tehtävätyyppejä, toinen alaluku keskittyy palautteen laadulliseen luokitteluun ja viimeinen alaluku tekee yhteenvedon ohjelmakoodin arvioinnissa käytettäviin arvostelukriteereihin.

3.1. Tehtävätyypit

Ensimmäinen yleisesti esiintyvä tehtävätyyppi on monivalintakysymykset. Monivalintatehtävissä jokaiselle kysymykselle on ennalta määritetyt yksikäsitteiset oikeat vastaukset, jolloin automaattinen arviointi on suoraviivaista. Tehtävätyyppi soveltuu kuitenkin huonosti käsitteellisen ymmärryksen mittaamiseen (8). Hyvien monivalintakysymysten keksiminen on hankalaa, joten monivalintojen hyödyntäminen ohjelmointikursseilla on mahdollista vain pienissä määrin (3). Ala-Mutkan mukaan huolellinen kysymysten suunnittelu kuitenkin mahdollistaa käsitteellisen ymmärryksen mittaamisen monella tasolla (7) ja tehtävätyyppi onkin siten käytössä monissa oppimisjärjestelmissä (8).

Monivalintojen sijaan vertais- ja itsearviointi mahdollistaa joustavan arvioinnin, joka sietää pienet virheet ja huomioi ratkaisujen laadun. Myös vertaisarviointi skaalautuu suurille kursseille, sillä jokainen opiskelija tuottaa myös uuden arvioijan. Menetelmän kääntöpuolena on ohjelmoinnin aloittelijoille tyypillinen taipumus kehittää omia, pedagogisesti kyseenalaisia arviointiperusteita, joten arviointityön laadun takaaminen on vaikeaa (8). Vertais- ja itsearviointia kuitenkin pidetään arvokkaana oppimismahdollisuutena, joten arviointimenetelmänä se suosittu myös massiivisilla ohjelmointikursseilla (3).

Yksi automaattisen arvioinnin pääsuuntaus on kehittää ja hyödyntää erilaisia tarkastimia. Tarkastimet ovat oppimisjärjestelmästä erillisiä ohjelmia, jotka ovat vastuussa arviointityön tekemisestä. (8) Tarkastimia on suuri joukko erilaisia, mutta niissä hyödynnettävät arviontimenetelmät voidaan jakaa kahteen pääkategoriaan – staattiseen ja dynaamiseen analyysiin – joita käsitellään tarkemmin luvussa 4.

Automaattisen arvioinnin hyötyihin kuuluu nopea palautteen saaminen (8), joka voidaan saavuttaa monivalinnoilla ja tarkastimilla yhtä lailla. Lyhyt viive palautuksen ja palautteen saamisen välillä mahdollistaa useamman yrityksen, jolloin opiskelija voi oppia saamastaan palautteesta, korjata ratkaisunsa ja tehdä uuden palautuksen. (8)

3.2. Palautekategoriat

Arvioinnin kaksi tärkeintä tavoitetta ovat mitata opiskelijan edistymistä sekä tuottaa hyödyllistä ja motivoivaa palautetta opiskelijan palauttamille ratkaisuille (9). Tavoitteen mukaan palaute voidaan siten jakaa laadultaan summatiiviseen ja formatiiviseen (10). Summatiivinen arviointi keskittyy muuntamaan vastauksien oikeellisuutta arvosanoiksi siinä missä formatiivinen palaute tarkoittaa sanallista palautetta, joka selittää ratkaisua ja voi esimerkiksi osoittaa virhekäsityksiä.

Summatiivinen palaute mittaa opiskelijan edistymistä kurssilla ja siten toimiikin motivaation lähteenä opiskelijalle ja muodostaa perustan kurssiarvosanalle (7). Formatiivisella palautteella on kuitenkin tärkeä rooli oppimisen tukemisessa (8; 9).

3.3. Arvostelukriteerit

Opiskelijan kirjoittaman ohjelmakoodin arvioinnin kaksi pääkohtaa ovat toiminnallisuus ja tyyli. Toiminnallisuuden tarkastelu pyrkii määrittämään toimiiko opiskelijan ohjelma oikein ja tehtävänannon vaatimalla tavalla. Useimmat Ihantolan, Ahoniemen, Karavirran ja Seppälän tutkimuksessaan tarkastelemat kaupalliset oppimisjärjestelmien tarkistimet nojasivatkin toiminnallisuustesteihin (7). Lähestymistavan suosio ei ole ihme, sillä se tuottaa luotettavan vastauksen siihen vastaako opiskelijan ohjelma tehtävänannossa kuvattuja vaatimuksia ohjelman toiminnallisuudelle.

Ohjelmoinnin luonteelle tyypillistä on, että jokaiseen ongelmaan on olemassa ennalta määrämätön määrä oikein toimivia vastauksia. Ratkaisun oikeellisuus ei kuitenkaan millään tavoin takaa ratkaisun laatua, joten arvostelun täydentämisessä käytetään usein apuna tyylintarkastusta. Tyylintarkastuksen ideana on arvioida ratkaisun laatua tutkimalla sen monimutkaisuutta ja kuinka usein koodissa esiintyy huonoja käytänteitä kuten käyttämättömiä muuttujia. Huonosta tyylistä tehdään pistevähennyksiä. (8) Tyylintarkastuksen automatisointi on kuitenkin vaikeaa, sillä minkä tahansa yksittäisen koodinpätkän mielekkyys riippuu vahvasti sen kontekstista ja käyttötarkoituksesta. (3)

Sallittujen palautusten määrä on myös eräs keskustelun aihe. Jotkin asiantuntijat vannovat rajoittamattomien palautuskertojen nimeen, sillä se mahdollistaa ratkaisun työstämisen iteratiivisesti vähän kerrallaan. (3) Toiset suosittelevat rajoittamaan palautuskertoja, sillä he katsovat rajoittamattomien palautuskertojen kannustavan väärinkäyttämään testausjärjestelmää sen sijasta, että opiskelijat opettelisivat testaamaan koodiaan itse (2).

4. LÄHESTYMISTAVAT TARKASTIMIEN TOTEUTTAMISEEN

Tarkastimet ovat se osa oppimisjärjestelmää, jonka vastuulla on tuottaa palautetta ja arvostelu ratkaisuille, joita opiskelijat palauttavat oppimisjärjestelmään. Tässä luvussa tehdään yhteenveto yleisesti käytetyistä lähestymistavoista tarkastimien toteuttamiseen. Ensimmäisessä aliluvussa lähestymistavat luokitellaan kahteen pääkategoriaan sen mukaan suoritetaanko palautettu ohjelmakoodi vai ei. Jälkimmäisessä aliluvussa käsitellään suosituimpia tarkastimissa käytettyjä testausstrategioita.

4.1. Lähdekoodin analyysin pääkategoriat

4.1.1. Staattinen analyysi

Staattinen analyysi käsittää kaiken analyysin, jossa tarkastettavaa ohjelmakoodia ei suoriteta. Lähestymistapa perustuu kirjoitetun koodin ominaisuuksien tutkimiseen. Eräs lähestymistapa staattisen analyysin työkalun toteutukseen on kuvattu Beyn, Jermannin ja Dillenbourgin paperissa. Algo+ -niminen työkalu vertailee kutakin palautettua ratkaisua yleisimmin esiintyviin samaan tehtävään palautettuihin ja arvioituihin ratkaisuihin. Opiskelijalle annetaan näkyviin sama palaute, jonka opettaja on kirjoittanut samankaltaisimmalle esiarvioidulle verrokkiratkaisulle tai jos vastaavuuksia ei löydy aikaisemmista palautuksista, lähetetään ratkaisu arvioitavaksi opettajalle. (8)

Staattisen analyysin huonosta maineesta tarkkuudessaan huolimatta paperissa esitetty lähestymistapa mahdollistaa palautteen ja osapisteiden antamisen jopa ratkaisuille, jotka eivät käänny tai kaatuvat suorituksen aikana (8). Täten opiskelijoiden on mahdollista saada palautetta ja osapisteitä melkein toimivasta ratkaisusta. Lisäksi opiskelijat saavat välitöntä apua tehtävän ratkaisemisen tueksi nähdessään opettajan antamaa palautetta ratkaisusta, jossa esiintyy yleisiä väärinkäsityksiä.

4.1.2. Dynaaminen analyysi

Dynaamiseksi analyysiksi luokitellaan kaikki testaaminen, jossa arviointiprosessiin kuuluu arvioitavan koodin suorittaminen. Yleisin lähestymistapa dynaamiseen arviointiin on toteuttaa sarja ohjelman toiminnallisuutta testaavia testitapauksia, joiden avulla päätellään, toimiiko ohjelma vaaditulla tavalla. Osapisteitä ja palautetta annetaan yleensä jokaisesta testitapauksesta erikseen.

Lähestymistavan kääntöpuolena on vaatimus testattava ohjelmakoodin suoritettavuudelle, jotta se voi läpäistä testitapauksia. (8) Tarkastimien tulee varautua kääntymättömiin, kaatuviin sekä ikuisiin silmukoihin jääviin palautuksiin, jotta arvosteluprosessi voidaan edes suorittaa loppuun. Aloittelevat ohjelmoijat palauttavat usein ikuiseen silmukkaan jäävää tai muuten suoritukseltaan päättymätöntä koodia, jonka suoritus arviointijärjestelmässä tulee aikakatkaista. (10) Täysin toimimattomalle koodille on mahdotonta tuottaa muuta palautetta kuin että sitä ei voi suorittaa. Myös ohjelmakoodin kompleksisuuden arviointi on dynaamisen analyysin työkaluilla hankalaa, mikä sen sijaan staattisen analyysin keinoin onnistuu helposti. (8)

Palautetun koodin suorittaminen myös altistaa arviointijärjestelmän pahansuovalle tai muutoin vaaralliselle koodille. Yleisesti käytettyjä keinoja turvallisuuden takaamiseksi ovat pääsynesto tunnistamattomilta käyttäjiltä, arviointiprosessin eristäminen kontteihin tai virtuaalikoneisiin sekä Linux-käyttäjäryhmien hyödyntäminen. Turvallisuus tunnistautumisessa ja käyttäjäryhmien hyödyntämisessä perustuu siihen, ettei opiskelijat ohjelmoinnin alkeiskursseilla yleensä omaa riittävää tietotaitoa järjestelmän hyväksikäyttöön tarkoituksella. (10) Testattavan koodin suorittamisen eristäminen konttiin[1] tai virtuaalikoneeseen puolestaan estää arviointiprosessia aiheuttamasta harmia isäntäjärjestelmälle, sillä ongelmat pysyvät eristyksissä kertakäyttöisen arviointiympäristön sisällä.

4.2. Toiminnallisuuden testaaminen

Ohjelman toiminnallisuuden testaamiseen on olemassa useita lähestymistapoja. Useimmiten testeissä yhdistellään useampaa testausstrategiaa. Tässä alaluvussa tehdään yhteenveto näistä lähestymistavoista.

4.2.1. Yksikkötestit

Yksikkötestit määrittelevät testitapauksia, joista jokainen tarkastelee pientä osaa ohjelman toiminnasta. Testitapaus voi tarkastella esimerkiksi yksittäisen metodin toimintaa määrittääkseen toteuttaako metodi vaaditun rajapinnan oikein. Tämä voidaan tehdä kutsumalla metodia luodulle oliolle ja tutkimalla saiko metodikutsu aikaan oikeanlaisen muutoksen olion tilassa, tai tutkimalla palauttaako aliohjelma oikean arvon sille annetuilla parametrien arvoilla.

---

[1] Konttivirtualisointi on tekniikka, jossa isäntäkäyttöjärjestelmän ominaisuuksia suoraan hyödyntämällä voidaan hallinnoida kontteja, joihin viedään ennalta määrätyn mallin pohjalta vain tarvittava määrä ohjelmistoa ja tiedostoja. Konttivirtualisointi on perinteisiä virtuaalikoneita kevyempi tekniikka, joka mahdollistaa konttien (engl. *container*) mielivaltaisen luomisen ja poistamisen automaattisesti. (11)

Monet ohjelmointikielien standardikirjastot tarjoavat ohjelmistokehyksiä (engl. *framework*) yksikkötestaamiseen, mikä helpottaa testien toteuttamista ja ajamista. Yksikkötestit ovat käytössä muun muassa Sveitsissä Lausannen liittovaltion teknillisen instituutin (*Ecole Polytechnique Fédérale de Lausanne*) MOOC-kursseilla (8).

Yksikkötestien kääntöpuolena on se, että niiden hyödyntäminen vaatii testattavassa koodissa käytettävän testeihin ennalta määriteltyjä tunnisteita (10). Jotta opiskelijan palauttaman koodin funktioita voidaan testata, tulee funktioiden nimet, parametrit ja parametrien järjestys vastata testeissä olevia, mikä voi olla epäintuitiivista aloittelevalle ohjelmoijalle.

4.2.2. Syöte/tuloste -testit ja syötevirtojen hallinta

Syöte/tuloste -testit, eli I/O-testit (engl. *input/output tests*) testaavat ohjelman toiminnallisuutta syöttämällä ohjelmalle ennalta määritetyt syötteet ja vertaamalla ohjelman tulosteita tunnetusti oikein toimivan malliohjelman tulosteisiin samoilla syötteillä. Mikäli testattava ohjelma ja malliohjelma tuottavat identtiset syötteet, päätellään testattavan ohjelman toimivan oikein. (10) Lähestymistavan etuihin kuuluu, että ohjelmasta voidaan testata mielivaltainen määrä ominaisuuksia yhdellä ohjelman suorituskerralla.

I/O-testauksessa hyödynnetään usein apuna syötevirtojen hallintaa (10) eli tekniikkaa, jossa ohjelman tulostevirta ohjataan omaan säiliöönsä standarditulostevirran sijasta. Säiliönä toimii syöte- tai tulostevirran tallentamiseen ja käsittelyyn soveltuva olio, jonka avulla syötevirtaa voidaan prosessoida ja verrata malliohjelman tulosteisiin. Ennen vertailua voidaan tulosteita prosessoida esimerkiksi poistamalla niistä välilyöntejä ja rivinvaihtoja, tai muuntamalla kirjaimia isoiksi tai pieniksi silloin, kun niillä ei ole tehtävän arvostelun kannalta merkitystä. Tämä mahdollistaa joustavamman arvostelun, kun jokainen puuttuva iso alkukirjain tai ylimääräinen välilyönti palautetun ohjelman tulostuksissa ei aiheuta testitapauksen hylkäämistä.

4.2.3. Regressiotestit

Regressiotestaaminen on tekniikka, jossa samoja testitapauksia ajetaan koodille pitkin kehittämisprosessia koodin mahdollisen rikkoontumisen estämiseksi kehittämisen sivutuotteena. Wilcoxin mukaan uusimmat nykyaikaiset automaattiseen testaamiseen kehitetyt kaupalliset ohjelmistokehykset soveltuvat huonosti sovellettavaksi opetuksessa, sillä ne perustuvat regressiotesteihin. Regressiotestaaminen on teollisen ohjelmakoodin validointiin kehitetty tekniikka, jolla on hyvin erilainen tavoite kuin pedagogisen arvon tuottaminen. (10)

5. ESIMERKKI TARKASTIMIEN HYÖDYNTÄMISESTÄ KURSSIKÄYTÖSSÄ

Tampereen yliopiston kurssitarjontaan kuuluu tekniikan kandidaattiopiskelijoille suunnattu kurssi Ohjelmointi 1: Johdatus. Kurssilla opetellaan ohjelmoinnin alkeita Python 3 -ohjelmointikielellä ja kurssimateriaalit julkaistaan verkkopohjaisesti Plussa-nimisessä oppimisympäristössä. Plussa on Tampereen yliopiston ilmentymä Aalto-yliopiston kehittämästä modernista verkkopohjaisesta oppimisjärjestelmästä nimeltä A+ (11).

Kurssin arvosana määräytyy oppimisjärjestelmään palautettavista viikkotehtävistä kerättävien pisteiden mukaan, vaikka kurssilla järjestetäänkin oppimista tukevia vapaaehtoisia luentoja ja suoritusvaatimuksiin kuuluu sähköinen tentti. Tehtävänannot löytyvät kurssialustalta Plussasta samaan tapaan kuin muukin kurssimateriaali. Tehtävien palauttaminen tapahtuu kirjautuneena Plussaan, joka lähettää vastauksen arvioitavaksi erilliselle arvostelutoiminnot sisältävälle palvelimelle. Palvelimella toimiva arviointijärjestelmä tuottaa HTML-muotoisen palauteen ja pistemäärän, jotka lähetetään arvostelupalvelimelta takaisin Plussaan, joka puolestaan näyttää palautteen opiskelijalle ja tallentaa saadut pisteet tietokantaan.

5.1. Konttitehtävät

Kurssin ohjelmointitehtävät ovat niin sanottuja konttitehtäviä, joiden arviointi tapahtuu muusta järjestelmästä eristetysti konttivirtualisoidussa ympäristössä. Tehtävien arvostelua konteissa hallinnoi erillisenä palveluna toteutettu arviointipalvelu, joka saa palautustiedostot anonyymeinä Plussalta ja joka palauttaa arvioinnin tulokset Plussalle. Konttitehtäville ominaista on, että niiden tarkastimet voidaan toteuttaa mielivaltaisesti. Ainoa arviointipalvelun asettama vaatimus on, että palautuksesta kerätty pistemäärä ilmoitetaan arviointijärjestelmälle sen vaatimassa muodossa ja mahdollinen muu palaute annetaan mielivaltaisena HTML-tiedostona, jonka arviointipalvelu välittää takaisin Plussaan näytettäväksi opiskelijalle. Käytännössä siis kurssin ohjelmointitehtävien tarkastimet voidaan toteuttaa vapaasti tarkastamaan juuri kunkin tehtävän kannalta oleellisia asioita.

Suurin osa kurssin konttitehtävistä on toteutettu siten, että tehtävän pisteet ja palaute generoidaan puhtaasti kontin sisälle rakennetussa tarkastimessa, jossa tarkastusprosessi on jaettu vaiheisiin järjestelmän jatkokehittämismahdollisuuksia ajatellen. Ensimmäisessä vaiheessa palautustiedostolle suoritetaan staattinen analyysi, jolla koodista et-

sitään laittomia avainsanoja, kuten `exit` tai `quit`. Palautusta ei oteta arvosteluun ollenkaan, mikäli opiskelija käyttää koodissaan kiellettyjä käskyjä, vaan niiden löytyessä arvosteluprosessin testien ajamisvaihe ohitetaan kokonaan. Palautukselle annetaan nolla pistettä ja opiskelijalle lähetettävään palautteeseen kirjataan koodin sisältäneen laittomia käskyjä.

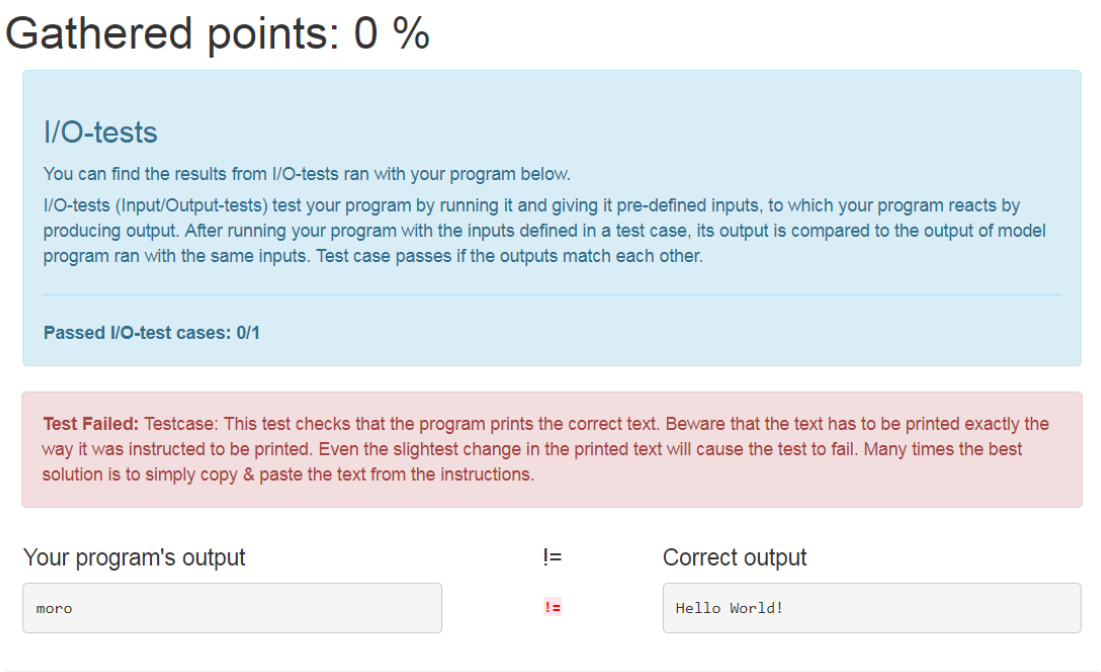
Arvosteluprosessin toisessa vaiheessa koodille suoritetaan tehtäväkohtaisesti määriteltyjä testejä, jotka määrittelevät toimiiko ohjelma annettujen vaatimusten mukaisesti. Toiminnallisuustestien tulokset tallennetaan väliaikaiseen tiedostoon prosessin seuraavaa vaihetta varten, jossa testien tulosten ja tehtävälle määriteltyjen tehtäväkohtaisten asetusten pohjalta muodostetaan palaute ja pistemäärä palautukselle. Prosessin viimeisessä vaiheessa palaute viedään HTML-muotoon, ja arviointijärjestelmälle annetaan pisteytyskäsky, joka lähettää palautteen ja pisteet Plussaan opiskelijan nähtäväksi. Lopuksi kontti poistetaan. Opiskelija pääsee tutkimaan palautetta ja tarvittaessa korjaamaan ratkaisuaan annetun palautteen pohjalta.

5.2. Toiminnallisuuden testaaminen konttitehtävissä

Toiminnallisuuden testaaminen perustuu I/O-testeihin ja yksikkötesteihin. I/O-testejä käytetään erityisesti kurssin alkupuolen tehtävissä testaamaan tulostaako ohjelma tehtävänannon määritelmän mukaisia tulostuksia annetuilla syötteillä. Jokaiselle tehtävälle on määritelty nollasta kymmenkuntaan I/O-testitapausta, joissa tarkastellaan ohjelman tärkeimpiä ominaisuuksia ja yleisimpiä virhetilanteita. Esimerkiksi Kivi-paperi-sakset -tehtävässä voitaisiin tarkastellaan osaako ohjelma päätellä voittajan oikein ja toisaalta osaako ohjelma käsitellä myös virheellisiä syötteitä.

I/O-testien soveltuvat myös pienten – esimerkiksi alle 10-rivisten – ohjelmien toiminnallisuuden testaamiseen hyvin, sillä I/O-testien ajaminen koodille ei vaadi Python-kielen tapauksessa varsinaisen pääohjelman olemassaoloa vaan riittää, että ohjelma voidaan suorittaa. Testit, jotka antavat ohjelmalle syötteitä ja tutkivat ohjelman tulosteita on katsottu kurssilla myös kurssiosallistujien näkökulmasta helppotajuiksi, sillä tapa on sama kuin minkä kurssilaiset ensimmäisenä oppivat ohjelman kanssa vuorovaikuttamiseen. Testit toimivat tällöin samaan tapaan kuin kurssilaiset.

Kurssin ensimmäisessä konttitehtävässä opiskelijan pitää toteuttaa Python-ohjelma, joka tulostaa tekstin: ”Hello World!”. Ohjelmalle ajetaan kontissa yksi I/O-testi, jonka tulostusta verrataan mallikoodin tuottamaan tulostukseen. Jos tekstit eroavat toisistaan, opiskelija saa nolla prosenttia tehtävän kymmenestä maksimipisteestä ja kuvassa 1 näkyvän palautteen tehtävästä.



Kuva 1: Kuvakaappaus opiskelijan käyttöliittymästä Plussassa, kun Moro maailma! -tehtävään palautetaan virheellisesti tulostava ohjelma.

Kuvan 1 mukaisesti alussa näkyy selitelaatikko I/O-tehtäville, jota seuraa listaus ohjelmalle ajetuista I/O-testitapauksista. Jokaiseen testitapaukseen liittyy omassa laatikossaan selite, jota seuraa vertailuosio. Vertailuosiossa opiskelijan ja mallikoodin suorituksen tulosteet näytetään rinnakkain ja keskisarakkeessa näkyy rivikohtaisen vertailun tulos. Symboleita ’!=’ ja ’==’ käytetään merkitsemään joko eri- tai yhtäsuuruutta sen mukaan vastaavatko rivit merkki merkiltä toisiaan. Myöhemmissä tehtävissä tulostukset sisältävät enemmän rivejä, jolloin rivikohtainen vertailu auttaa opiskelijaa paikantamaan virheen aiheuttavan rivin nopeammin. Tulostukset on aseteltu rinnakkain helpottamaan niiden silmämääräistä vertailua.

Kun kurssilla edetään muuttujista ja tavallisimmista ohjausrakenteista funktioihin, otetaan uutena testityyppinä käyttöön I/O-testien rinnalle yksikkötestit, jotka soveltuvat funktioiden – ja kurssilla myöhemmin myös luokkien – toiminnallisuuden testaamiseen paremmin kuin I/O-testit. Yksikkötestien hyödyntämisen kääntöpuolena ne asettavat rajoituksia opiskelijan ratkaisussa käytettäville tunnisteille – esimerkiksi yksikkötestattavan funktion nimen sekä sen parametrien määrän ja järjestyksen tulee vastata sitä otsikkoa, jonka testit olettavat sen olevat, tai muuten testikoodi ei osaa tunnistaa opiskelijan ratkaisusta oikeaa testattavaa funktiota.

5.3. Tehtäväkohtaiset asetukset

Tehtäville on kurssilla asetettu pääsääntöisesti kymmenen sallittua palautuskertaa, joiden puitteissa opiskelijan tulisi saada aikaiseksi tehtävänannon mukaisesti toimiva ohjelma. Automaattisten tarkastimien käyttäminen puoltaa mahdollisuutta useampaan palautuskertaan, sillä edes loogisesti oikein toimiva ohjelma ei välttämättä läpäise testejä ensimmäisellä kerralla, jos vaikkapa kaksi kirjainta tulosteessa ovat vaihtaneet paikkaa. Tämä johtuu siitä, että I/O-testit vertailevat opiskelijan ohjelman ja malliohjelman tulosteita merkkikohtaisesti.

Konttitehtävien I/O-testeissä kuitenkin hyödynnetään syöte- ja tulostusvirtojen hallintatekniikkaa. Tekniikalla I/O-testeihin on toteutettu ominaisuus, jolla opiskelijan ja malliohjelman tulosteista voidaan poistaa tyhjät merkit ja muuntaa tulosteet suuraakkosiksi ennen vertailua. Tämä vähentää niiden kirjoitusvirheiden määrää, joihin merkkikohtainen vertailu jää kiinni. Riippuen tehtävän luonteesta, kurssin vetäjä voi valita tehdäänkö vertailu käsitellyille vai käsittelemättömille tulostuksille. Moro Maailma! -tehtävässä tavoitteena oli oppia tulostamaan, joten tulostettavalla tekstillä ei sinänsä ole pedagogista merkitystä. Sen sijaan tehtävässä, jossa opetellaan tulostamaan 10 merkin levyiseen tekstikenttään, on mielekästä tarkistaa, että myös välilyönnit tulostuvat oikein.

Myös kurssilla Ohjelmointi 1: Johdatus on havaittu useiden palautuskertojen rohkaisevan opiskelijoissa käytöstä jättää ohjelman testaaminen automaattitestien vastuulle sen sijaan, että opiskelijat testaisivat ratkaisujaan itse. Kurssin tavoitteisiin kuitenkin kuuluu oppia testaamaan omaa ohjelmakoodia, joten konttitehtäviin on kehitetty mekanismeja, jotka pakottavat opiskelijan testaamaan ratkaisujaan itsenäisesti.

Esimerkiksi I/O-testejä sisältäville tarkastimille on mahdollista valita, näytetäänkö palautteessa testitapauksen selitteen perässä ollenkaan vertailuosiota, jossa näkyy kaikki syötteet ja tulostukset, joita testitapauksen ajon aikana konsoliin ilmaantuu. Jos vertailuosiota ei näytetä, jää opiskelijan vastuulle selvittää millaisella syötteellä ja miten ohjelma toimii väärin. Myös yksikkötestattavissa tehtävissä on mahdollista näyttää joko yleinen kirjallinen kuvaus testitapauksesta tai suoraan millä syötearvoilla mikäkin funktio palauttaa odotetusta poikkeavan tuloksen.

Viimeinen jokaiselle tehtävälle erikseen säädettävissä oleva asetus määrittää voiko tehtävästä saada osapisteitä, vai pitääkö ratkaisun läpäistä kaikki testitapaukset, jotta siitä voi saada pisteitä. Suuremmissa projektitehtävissä voi olla mahdollisuus esimerkiksi kerätä bonuspisteitä, joilla projektin arvosanaa voi korottaa, mutta joiden toteuttamista ei voida vaatia kaikilta opiskelijoilta.

6. ARVIO TARKASTUSKÄYTÄNTÖJEN TOIMIVUUDESTA TARKASTELTAVASSA JÄRJESTELMÄSSÄ

6.1. Järjestelmäratkaisuista saatavat edut

Plussan käyttämisestä kurssialustana kurssilla *Ohjelmointi 1: Johdanto* saadaan oppimisjärjestelmille tyypillisiä hyötyjä. Arvostelu on käytännössä välitöntä, sillä automaattiset tarkastimet tuottavat palautteen opiskelijan nähtäville pääsääntöisesti joissakin kymmenissä sekunneissa palautuksesta. Kerran toteutetulla tehtävätarkastimella voidaan tarkastaa samanaikaisestikin useita palautuksia kerrallaan ja tehtävätarkastin voidaan kierrättää kurssin seuraavalla toteutuskerralla ilman lisäinvestointeja. Opiskelijat saavat kaikista tekemistään palautuksista palautteen, josta selviää toimiiko ratkaisu kuten on tarkoitettu.

Järjestelmä taipuu monipuoliseen, kattavaan ja joustavaan toiminnallisuuden tarkastamiseen. Jokaiselle tehtävälle voidaan luoda sopiva määrä testitapauksia, jotka voivat testata ohjelman syötteenlukua, tulosteiden oikeellisuutta ja myös rajapintojen toiminnallisuutta. Myös tehtävistä annettava formatiivinen palaute on pitkälle muokattavissa tehtävän tarpeita vastaavaksi. Tehtävän luonteen mukaan epäonnistuneista testitapauksista voidaan näyttää pelkästään kyseiseen testitapaukseen liittyvä selite tai myös vertailuosio, josta on suoraan nähtävissä millaisessa tilanteessa ohjelma toimii väärin. Palaute myös kohdentuu suoraan ongelmakohtaan, mikä helpottaa ongelmien paikantamista.

Automaattinen arviointi myös helpottaa arvostelua ja tekee siitä tasapuolista ja läpinäkyvää. Opettajan näkökulmasta arvostelutyötä on vähemmän ja kurssi skaalautuu helposti suurellekin opiskelijamäärälle. Oppilaan näkökulmasta arvostelu on reilua, sillä jokainen palautus arvostellaan keskenään samoilla kriteereillä ja pisteitä saa tehtävänannossa kuvatun toiminnallisuuden toteuttamisesta. Opiskelija voi seurata edistymistään kurssilla keräämiensä pisteiden perusteella, sillä pisteet muuntuvat suoraan kurssiarvosanaksi.

Tehtävien tarkastaminen konttivirtualisoidussa ympäristössä onnistuu eristämään mahdollisen vaarallisen koodin suorittamisen ympäristöön, jossa siitä ei ole haittaa muulle järjestelmälle.

6.2. Teknisten ratkaisujen kehittämiskohdat

Järjestelmän teknisissä ratkaisuissa on kuitenkin myös kehittämisen varaa. Järjestelmän ominaisuuksista puuttuu kokonaan automaattinen ohjelmointityylin tarkastaminen. Vaikka palautetta ohjelmointityylistä saadaan opetushenkilökunnan käsin arvostelemista projekteista, tyylintarkastus olisi mahdollista toteuttaa myös koneellisesti, jolloin henkilöresursseja voitaisiin kohdentaa muihin tehtäviin.

Esitarkastusvaiheessa tehtävä laittomien avainsanojen tarkastus sisältää vain ohjelmasta poistumisen käyttämällä exit-toiminnallisuutta. Poistumiskieltoa perustellaan sillä, että yksikään kurssin tehtävä ei vaadi poistumista muualta kuin pääohjelmasta palaamalla ja kurssilla opetellaan selkeiden ratkaisujen tuottamista myös ohjelman suoritusjärjestyksen kannalta, eikä esimerkiksi poistumisen edellyttämä siivoustoimenpiteistä huolehtiminen kuulu alkeita opettelevan ohjelmoijan tietotaitoihin.

Esitarkastuksen avainsanojen tunnistusmekanismi mahdollistaisi kuitenkin myös laajemman avainsanojen tarkastelun. Esimerkiksi `import`-käskyjen lisääminen kiellettyjen käskyjen listalle mahdollistaisi sisäänrakennettujen Python-moduulien avulla tapahtuvan oppimistavoitteiden kiertämisen tarkastamisen. Esimerkiksi tehtävässä, jossa opetellaan määrämuotoisen tiedon lukemista tiedostosta tietorakenteeseen, on sama toiminnallisuus helppo saavuttaa hyödyntämällä Python-kieleen sisäänrakennetun `JSON`-moduulin `load`-metodia. Siinä missä tiedon hakeminen Internetistä on ohjelmoijalle tärkeä taito, on ohjelmoijan syytä omata myös käsitys tiedostojen käsittelemisen perusteista. Ymmärryksen olemassaolosta ei voida varmistua niiden ratkaisujen osalta, joissa toiminnallisuus on toteutettu ohjelmointikielen valmiita toiminnallisuuksia hyödyntämällä ja siksi niiden ratkaisujen hylkääminen voisi olla perusteltua.

6.3. Havaintoja järjestelmän käytöstä

Myös kurssilla *Ohjelmointi 1: Johdanto* on törmätty automaattiseen tarkastamiseen liittyvään ilmiöön vaatimuksesta kuvata ratkaisulta vaadittava toiminnallisuus tehtävänannossa yksityiskohtaisesti. Esimerkiksi murtolukuja käsittelevän tehtävän tehtävänannon täytyy kuvata mitä tapahtuu, kun käyttäjä syöttää nimittäjälle arvoksi nollan. Mikäli tehtävänanto ei ole yksiselitteinen, joutuu opiskelija arvaamaan mitä tehtävässä haetaan. Myös palautuskertoja voi kulua turhaan opiskelijan selvittäessä millaista toiminnallisuutta testit vaativat ratkaisulta, eikä tämäntyyppinen takaisinmallinnus ole tarkoituksenmukaista toimintaa ohjelmoinnin johdantokurssilla. Epämääräisesti kuvatun toiminnallisuuden takia epäonnistuvat testit aiheuttavat turhautumista, jos opiskelijalla ei ole mahdollisuutta selvittää miten ratkaisun odotetaan toimivan.

Toisaalta niin ikään liian pitkät tehtävänannon aiheuttavat turhautumista, sillä yksityiskohdat hukkuvat helposti tekstinpaljouteen, jolloin niiden toteuttaminen helposti unohtuu. Usein Kooditoriossa eteen tulevat kysymykset koskevatkin testejä, jotka ovat epäonnistuneet, koska opiskelijan ratkaisu käsittelee jonkin tehtävänannossa mainitun erityistapauksen toisella tavalla kuin mitä tehtävänannossa on ohjeistettu. Eräänlaisen osaratkaisun ongelmaan saa kuitenkin testitapauskohtaisista selitelaatikoista, joihin opettaja voi sisällyttää vihjeen esimerkiksi siitä, mikä on testitapauksen epäonnistumisen yleisimmin aiheuttava ongelma. Selitelaatikko näkyy opiskelijalle testitapauksen epäonnistuessa.

Toinen kurssilla niin ikään havaittu ilmiö on opiskelijoiden taipumus hyödyntää ohjelmakoodiansa ainoastaan automaattisilla tarkastimilla. Käyttäytymistä yritetään kurssilla ehkäistä piilottamalla tarkka virhepaikka tarkastimien antamasta palautteesta ja toisaalta muistuttamalla siitä, että omaa koodia täytyy opetella testaamaan, mutta lähestymistapa on havaittu puutteelliseksi. Tarkan virhekohdan piilottaminen aiheuttaa opiskelijoissa jonkin verran turhautumusta enemmän kuin se kannustaa kehittämään omia testejä. Kurssilla ei varsinaisesti opeteta muodollista oman koodin testaamista kuin keksimällä omia käsin ajettavia testitapauksia, jolloin opiskelijat helposti jättävät ei-pakollisen työn tekemättä. Testitapausten kirjoittamisen vaatiminen osana esimerkiksi projektitehtäviä voisi motivoida opiskelijoita paremmin opettelemaan myös muodollista testaamista.

Kurssin tehtäville annettu kymmenen palautuskertaa mahdollistaa ohjelman toiminnallisuuden kehittämisen ja testaamisen pala kerrallaan, mutta toisaalta se ei kuitenkaan salli päämäärätöntä työskentelyä. Palautuskerrat kuluvat loppuun nopeasti, jos palautuksia tehdään ilman mitään ajatusta siitä, miten koodia tulisi kehittää palautusten välissä. Syötevirtojen käsittelymekanismin hyödyntäminen tarkastuksissa vähentää jonkin verran kirjoitusvirheistä aiheutuvia turhia ratkaisujen hylkäyksiä, mikä säästää palautuskertoja ja parantaa automaattisten tarkastimien toteutuksen tarkoituksenmukaisuutta. Toisaalta myös virheenetsinnän tarkkuuden parantaminen merkkiin rivin sijasta voisi nopeuttaa opiskelijan työskentelyä mekaanisen virheenetsinnän osalta. Esimerkiksi puuttuvan välilyönnin tai väärän välimerkin löytäminen riviltä tekstiä on Kooditoriossa havaittu yllättävän hankalaksi. Virheellisten merkkien etsimisessä ei myöskään ole pedagogista arvoa, joten tehtävän voisi automatisoida.

6.4. Kurssin oppimismenetelmät

Kurssin oppimismuodot painottuvat aktiivisiin oppimismenetelmiin eli tekemällä oppimiseen ja itsenäiseen työskentelyyn. Oppimismenetelmät tähtäävät syväoppimiseen ja vaativat opiskelijaa ottamaan vastuuta omasta oppimisestaan. Ilman ohjausta kurssilla

ei kuitenkaan jäädä. Opiskelijan työskentelyä rytmittää viikoittaiset palautusten määräajat, ja yksilöllistä ohjausta on tarjolla kurssille lanseerattujen Kooditoriopäivystysvuorojen muodossa. Kooditoriossa opetusassistentti päivystää läpi kurssin ennalta sovittuina aikoina ja auttaa mahdollisten ongelmakohtien selvittämisessä. Myös ryhmätyöskentely on kurssilla sallittua.

Kurssin automaattiset tarkastimet tuottavat palautetta vain ohjelman toiminnallisuudesta. Kuitenkin myös ohjelmointityylillä on merkitystä ohjelmointitaidon arvioinnissa. Siksi kurssin arviointia ja palautteenantoa joudutaan täydentämään opetushenkilökunnan käsin tarkastamilla ohjelmointiprojekteilla, joita opiskelija toteuttaa omista oppimistavoitteistaan riippuen kolmesta viiteen kappaletta kurssin aikana. Käsin tarkastettujen projektien arviointi keskittyy ratkaisun ohjelmointityyliin ja yleisperiaatteisiin, joita automaattisilla tarkastimilla on hankala arvioida. Ihminen sen sijaan pystyy konetta joustavammin arvioimaan ratkaisujen tarkoituksenmukaisuutta ja monimutkaisuutta; esimerkiksi toisteinen koodi katsotaan lähtökohtaisesti huonoksi ohjelmointikäytännöksi, mutta erityisesti asetus- ja hakumetodien kohdalla on monesti järkevämpää sietää jonkin verran toisteisuutta kuin toteuttaa monimutkaisia kiertotapoja toisteisuuden välttämiseksi. Tarkoituksenmukaisen automaattisen toisteisuustarkastimen toteuttaminen voi siten olla vaikeaa siinä missä opetusassistentti puolestaan pystyy joustavaan kontekstisidonnaiseen arviointiin.

7. JOHTOPÄÄTÖKSET

Työssä tutkittiin lähestymistapoja automaattisten tarkastimien toteuttamiseen ohjelmoinnin alkeiskurssien opetuskäytössä. Työssä tehtiin kirjallisuuskatsaus siihen, mitä ovat modernit oppimisjärjestelmät, millaisia oppimismenetelmiä, tehtävätyyppejä ja arviointitapoja niissä voidaan hyödyntää ja millaista tukea oppimiseen ja opetukseen ne voivat tarjota. Lisäksi työssä arvioitiin tarkastimien toteuttamiskäytäntöjä.

Oppimisjärjestelmissä usein esiintyviksi tehtävätyypeiksi tunnistettiin monivalintatehtävät, ratkaisujen vertais- ja itsearviointi, sekä erilaiset automaattiset tarkastimet, joita työssä tarkasteltiin laajemmin. Sekä dynaamiseen että staattiseen analyysiin perustuvat lähestymistavat todettiin soveltuvan ohjelmointitehtävien arviointiin sekä formatiivisen ja summatiivisenkin palautteen tuottamiseen, vaikka dynaamisen analyysin lähestymistapa vaikutti olevan useammin oppimisjärjestelmissä hyödynnettävä lähestymistapa. Työ ei sisällä täydellistä listausta erilaisista lähestymistavoista arvioinnin toteuttamiseen ohjelmointikursseilla, vaan keskittyy syöte/tuloste -testeihin automaattiseen ja yksikkötestaamiseen.

Työssä tarkasteltiin Plussa-oppimisjärjestelmään toteutettua ohjelmoinnin johdantokurssia tehdyn kirjallisuusselvityksen pohjalta. Järjestelmän toteutus todettiin keskittyvän palautettujen ratkaisujen toiminnallisuuden tarkastamiseen ja teknisenä ratkaisuna järjestelmässä hyödynnettiin pääasiassa yksikkötestaamista ja syöte/tuloste -testejä. Tarkastinjärjestelmässä hyödynnetään syötevirtojen hallintatekniikkaa tarkastuksen mielekkyyden parantamiseksi ja tehtävien arvioinnilla konttivirtualisoidussa ympäristössä vaarallinen palautettu koodi saadaan eristettyä muusta oppimisjärjestelmästä. Toteutettu järjestelmä siis soveltuu ohjelmoinnin opetukseen.

Kurssin tarkastinjärjestelmän käytöllä saavutetaan oppimisjärjestelmien tärkeimpiä etuja – arviointityön automatisointi vähentää arviointiin liittyvää työkuormaa, opiskelijat saavat enemmän yksilöllistä ja välitöntä palautetta ja kurssi skaalautuu suurillekin opiskelijamassoille. Järjestelmästä tunnistettiin kuitenkin myös kehittämisen varaa, sillä esimerkiksi puuttuvan tyylitarkastuksen takia järjestelmä ei riitä ainoaksi palautteen ja arvioinnin lähteeksi, vaan arviointia joudutaan täydentämään käsin tehtävällä arvioinnilla. Myös oman ohjelmakoodin testaamiseen kannustamista voisi järjestelmässä kehittää.

LÄHTEET

[1] S. Gupta, S. K. Dubey, Automatic Assessment of Programming assignment. Department of Computer Engineering, Shri G. S. Institute of Technology & Science. Indore, Madhya Pradesh, Intia: Department of Computer Engineering Shri G. S. Institute of Technology & Science. 2012. 10.5121/csit.2012.2129.

[2] K. Ala-Mutka, A Survey of Automated Assessment Approaches for Programming Assignments. Tampere: Tampereen teknillinen yliopisto. 2005.

[3] V. Pieterse, Automated Assessment of Programming Assignments. University of Pretoria. s.l.: University of Pretoria. 2013.

[4] G. Conole, B. Warburton, A review of computer-assisted assessment. ALT-J, Research in Learning Technology, painos 13, numero 1. Southampton: University of Southampton. 2005, pp. 17–31.

[5] O. Hyppönen, S. Lindén, OPETTAJAN KÄSIKIRJA – OPINTOJAKSOJEN RAKENTEET, OPETUSMENETELMÄT JA ARVIOINTI. Espoo: Teknillinen korkeakoulu, Opetuksen ja opiskelun tuki. 2009. Osa/vuosik. 4/2009.

[6] S. Lonn, S. D. Teasley, Saving time or innovating practice: Investigating perceptions. Computers & Education. Ann Arbor: School of Information and USE Lab, University of Michigan. 2009. Osat/vuosik. 53/3, pp. 686–694.

[7] P. Ihantola, T. Ahoniemi, V. Karavirta, O. Seppälä, Review of recent systems for automatic assessment of programming assignments. Koli: s.n., 2010. Proceedings of the 10th Koli Calling International Conference on Computing Education Research.

[8] A. Bey, P. Jermann, P. Dillenbourg, A Comparison Between Two Automatic Assessment Approaches for Programming: An Empirical Study on MOOCs. 2018. Journal of Education Technology & Society Vol. 21. pp. 257–272.

[9] C. Sandeen, Assessment's Place in the New MOOC World. 2013. Journal of Research & Practise in Assessment vol. 8.

[10] C. Wilcox, Testing Strategies for the Automated Grading of Student Programs. s.l.: ACM. 2016. Proceedings of the 47th ACM Technical Symposium on computing science education. pp. 437–442.

[11] Cloud Academy Inc. CloudAcademy.com - Blog/Amazon Web Services, Container Virtualization: What Makes It Work So Well? [Verkossa] CloudAcademy Inc., 27.9.2015. [Viitattu: 11.4.2019.] https://cloudacademy.com/blog/container-virtualization/.

[12] Aalto yliopisto, A+ LMS. The extendable learning management system. [Verkossa] Aalto Yliopisto. 2019. [Viitattu: 4.2.2019.] https://apluslms.github.io/.